Scaling Test-time Compute for LLM Agents

Anonymous ACL submission

Abstract

Scaling test-time compute has shown remarkable success in improving the reasoning abilities of large language models (LLMs). In this work, we conduct the first systematic exploration of applying test-time scaling methods to language agents and investigate the extent to which it improves their effectiveness. Specifically, we explore different test-time scaling strategies, including: (1) parallel sampling algorithms; (2) sequential revision strategies; (3) verifiers and merging methods; (4) strategies for diversifying rollouts. We carefully analyze and ablate the impact of different design strategies on applying test-time scaling on language agents, and have the following findings: 1. Scaling test time compute could improve the performance of agents. 2. Knowing when to reflect is important for agents. 3. Among different verification and result merging approaches, the list-wise method performs best. 4. Increasing diversified rollouts exerts a positive effect on the agent's task performance. All code are available at https://anonymous.4open.science/r/ATTS-D74F.

1 Introduction

Language agents demonstrate exceptional capabilities in various domains (Tang et al., 2025; Zhou et al., 2023b; Hong et al., 2023; Face, 2025; Zhou et al., 2023c, 2024). For example, LangChain(repository, 2022) connects LLMs with various tools to solve different tasks in an end-to-end manner, while Meta-GPT(Hong et al., 2023) enables multiple AI Agents to take on different roles and collaborate to accomplish tasks. Recently, long-thinking models like O1 (Jaech et al., 2024) and R1 (Guo et al., 2025) showcase excellent reasoning abilities of Large Language Models (LLMs). Recent approaches (Li, 2025; Liang et al., 2025) leverage the extended thinking capabilities of long-activation models for planning, code writing, tool calling, and completing complex tasks. However, despite the strong capabilities of LLMs, they still struggle to match human performance in complex search and reasoning tasks (Zhou et al., 2023a; Koh et al., 2024a). This occurs due to remaining limitations in model capabilities, errors in task planning and question answering, and issues with complex tool calling abilities.

Increasing computational resources during the inference phase greatly enhances LLMs' performance. Some works (Liu et al., 2025a,b) improve model exploration during inference through different sampling strategies, achieving excellent scores in challenging areas like mathematical reasoning. (Snell et al., 2024) investigate the effects of scaling test-time computate, while (Xiong et al., 2025) focus on enhancing model performance through self-correction methods. However, directly applying TTS methods to the Agentic Framework presents many challenges. Unlike LLMs that solve specific problems in an end-to-end manner, agents typically decompose complex problems into distinct steps, invoking multiple models sequentially for resolution. Due to the extended sequence of steps and the accumulation of errors, traditional TTS methods (e.g., BoN) can significantly undermine the final outcome, because they randomly generate N responses at each step.

To address the aforementioned challenges, we first conduct a systematic exploration of test-time scaling methods for language agents. First, we investigate the effectiveness of different **parallel sampling** methods for agentic test-time scaling, including variants of Best-of-N (BoN), beam search, and tree search algorithms. We adapt and implement these parallel sampling mechanisms within language agents and showing that despite simplicity, BoN achieves the optimal performance. Subsequently, we investigate the effectiveness of various **sequential revision** techniques, such as reflection and self-refinement, for language agents. We introduce a reflection agent to summarize and reflect

based on the current state and recent actions/observations to help the agent consistently progress toward accomplishing the task. Experimental results show that the direct gains from having the agent perform reflection at each step are not obvious. Instead, allowing the agent to perform reflection when it performs poorly in the current step brings certain benefits. This indicates that **knowing when the agent should reflect is more important than having the agent perform reflection at every step directly**. Finally, we conduct a detailed study on the impact of different verify and result merging methods, including voting (selection of the majority from all candidates), scoring (selection based on scores), and list-wise (comparative selection of the optimal item from candidates) approaches. Our experimental results demonstrate that whether for merge results methods or verify methods, using the list-wise method outperforms other methods. This provides an effective verification method reference for agentic frameworks. Finally, we test different strategies to expand the agent's exploration space and enhance the diversity of different rollouts, and propose a multi-agent collaborative sampling strategy. Experimental results indicate that performance under multi-agent collaboration surpasses that of a single agent.

Our core contributions are:

- We explore the application of different parallel sampling strategies in agentic frameworks. Through parallel sampling strategies, agent performance can be significantly improved.
- We study the impact of sequential revision techniques in agentic frameworks. In particular, we point out that it is very important for agents to know when they should perform revision.
- We also conduct detailed comparative analysis of different verify and result merge strategies. Experiments show that the list-wise method significantly outperforms other methods.

2 Agentic Test-Time Scaling

In this section, we describe and compare different strategies for agentic test-time scaling, including: (1) Parallel Sampling Algorithms; (2) Sequential Revision Strategies; (3) Verifier and Result Merging Methods; (4) Strategies for Diversifying Rollouts.

2.1 Parallel Sampling Algorithms

To establish a comprehensive evaluation framework for our proposed methods, we regard several parallel sampling algorithms that are commonly used in the test-time scaling (TTS) domain as baselines.

Best-of-N (BoN)(Faria and Smith, 2025): Give a sample times N and question Q, the Best-of-N (BoN) method samples N independent responses from the LLMs:

$$\{R_1, R_2, ..., R_N\} = \text{BoN}(Q),$$

Then selects the best answer by verifier. The effectiveness of BoN relies heavily on the quality of the reward model and the diversity of sampled candidates.

Step-wise-Best-of-N (BoN-wise)(Wu et al., 2024a): BoN-wise generates N responses at each step. Specifically, given the thoughts $\{T1, T2, ..., T_{t-1}\}$ from the previous step at time t, BoN-wise generate N responses:

$$\{R_1, R_2, ..., R_N\} = \text{BoN-Wise}(Q, T_1, T_2, ..., T_{t-1}),$$

Beam Search(Yao et al., 2023): Beam search maintains a fixed-size beam K at each step. Specifically, at time t, it generates N responses for each leaf nodes LN_i, then maintains the most suitable K responses that :

$$\{R_1, R_2, ..., R_K\} = \text{BeamSearch}(Q, LN_i)$$

Most promising partial solutions at each generation step. This algorithm prunes less promising candidates early in the generation process based on cumulative log-probabilities or reward scores.

DVTS (Diverse Verifier Tree Search) (Beeching et al., 2024a): DVTS decomposes the task into K subtrees, where each subtree operates as an independent beam search algorithm. By exploring multiple subtrees in parallel, DVTS achieves more diverse search behavior. Under the same computational budget, it finds higher-quality solutions more effectively than a single beam search with deep exploration.

$$\{R_1, R_2, ..., R_K\} = \text{DVTS}(Q, SubTree_i)$$

2.2 Sequential Revision Strategies

Besides, given the previous steps $\{T_1, T_2, ..., T_{t-1}\}$ at time t, we leverage a reflection model RefM, to summarize information:

$$Sum_t = \text{RefM}(T_1, T_2, ..., T_{t-1}).$$

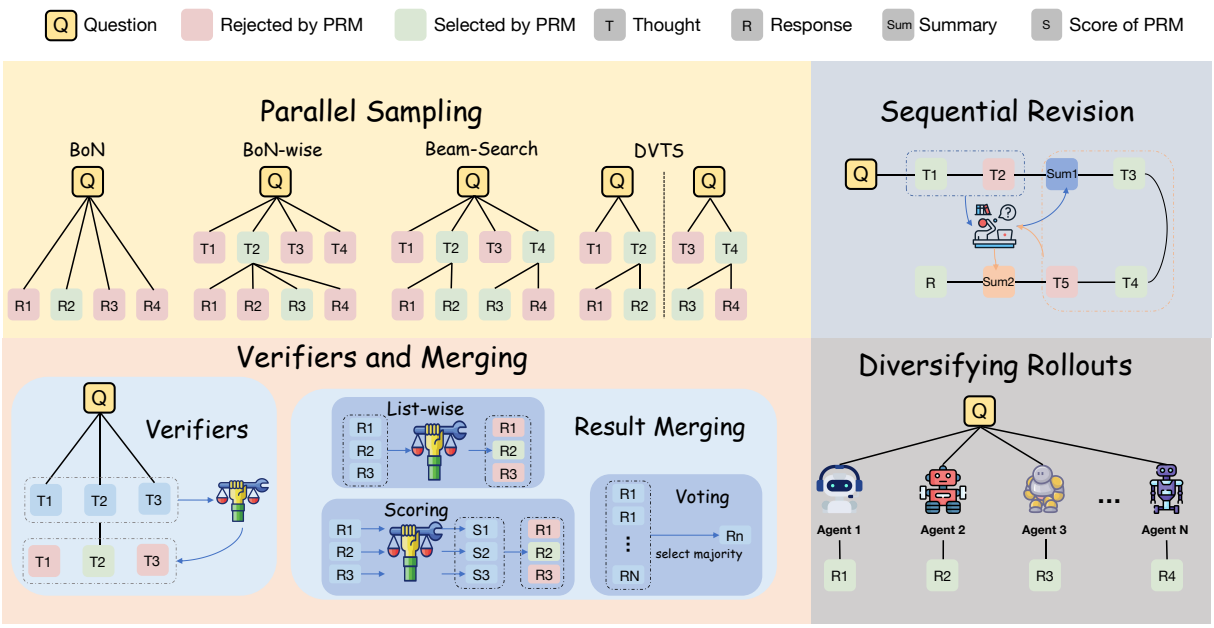

Figure 1: Overview of our agentic test-time scaling framework with four key strategies: (1) **Parallel Sampling**: BoN, BoN-wise, Beam Search, and DVTS;(2) **Sequential Revision**: Reflection model with threshold-driven re-generation;(3) **Verifiers and Result Merging Methods**: Scoring, list-wise, and majority voting;(4) **Diversifying Rollouts**: Sampling across heterogeneous agents.

To ensure the model understands when reflection is needed, we use a verify model to objectively score each step of the model to represent the quality of the current step action, and set different score $Threshold$. If and only if the model action score is less than the $Threshold$, the Sum_t is added into the LLM to generate the responses for time t. The prompt for Sequential Revision can be found in Appendix A.

2.3 Verifiers and Merging Methods

Verifiers To enable agents to receive positive feedback signals during the sampling process, we have designed two different process-based reward functions that evaluate the value of each sampling action.

scoring PRM: We score each thought steps at each intermediate step t to revise the final response. For N thought steps $\{T_1, T_2, ..., T_N\}$ generated at step t, we utilize a LLM as Reward Model (RM), to obtain the score of each response $S_i = RM(T_{current\_i})$. The prompt for scoring PRM can be found in Appendix A.

list-wise PRM: Another commonly used verify method is to select the optimal trajectory through direct comparison. For N thought steps $\{T_1, T_2, \ldots, T_N\}$ generated at step t, we provide all candidate actions to the LLM, asking it to select the optimal trajectory from among them, $S_i = \text{RM}(T_{\text{current\_i}})$. The prompt for list-wise PRM can be found in Appendix A.

Result Merging Methods We compare mainstream Result Merging approaches(Wang et al., 2024), including **voting**: Directly select the majority from all candidates, **scoring**: using verify for direct scoring, and **list-wise**: where the model directly selects the optimal answer from candidate responses.

2.4 Diversifying Rollouts

The efficiency of Parallel Sampling Algorithms is influenced by diversifying rollouts—more diverse rollouts mean the agent has a greater chance of exploring and discovering the correct answer. LLMs generate diverse candidates by controlling hyperparameters such as temperature and top_p.

However, in agent frameworks, employing multi-agents to collaboratively accomplish the same task often enhances task performance. To further increase diversity in the agents' sampling process, we utilize different LLMs as rollout models. Different LLMs often exhibit distinct capability profiles; some excel in coding, while others demonstrate exceptional performance in tool using. We have designed various agent combinations to maximize rollout diversity.

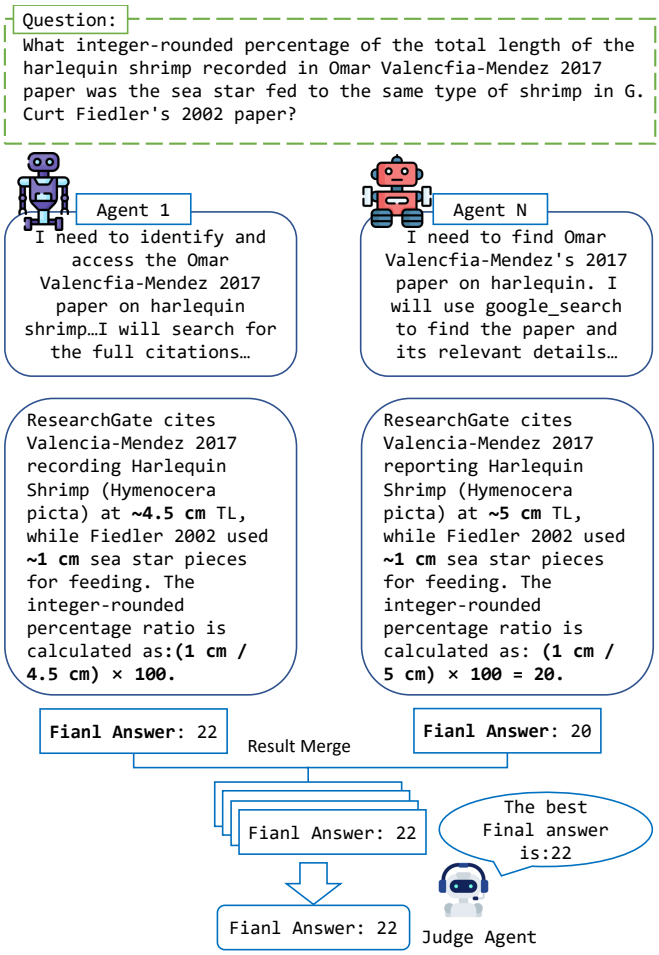

Figure 2: A case study: Given one question, the agent performs operations such as coding and tool calls during a single rollout, and returns diverse results. The judge agent will merge the final result and output the best answer.

3 Experiments

3.1 Experiments Setting

To thoroughly investigate the effects of TTS algorithms within the agentic framework, we conduct the following comparative experiments:

Comparison of Different Parallel Sampling Algorithms In order to comprehensively compare different parallel sampling algorithms in the agentic framework, we select mainstream parallel sampling algorithms, including BON, BON-wise, Beam-Search, Tree search, and conduct comparative experiments under identical experimental settings. We ensure a sampling width of 4, and for Beam-Search and DVTS, we fix the beam size at 2.

Comparison of Different Sequential Revision Strategies To investigate how self-reflection affects agent performance, focusing on when and how reflection should be applied. We set up the following two settings: **Step-based Reflection**: Reflection is conducted at every step to enable continuous error correction. **Score-based Reflection**: Initially, the Verify model scores each step of the agent's process. The agent performs reflection only when a step's score falls below a predefined threshold. To further explore how the frequency of reflection affects performance, we conduct ablation studies using three triggering thresholds: <8 (frequent), <5 (moderate), and <2 (selective).

Comparison of Different Verifiers and Merging Methods To investigate the impact of different Verifiers and Merging Methods. First, we compare the performance differences among three mainstream result merging methods, and then, based on the optimal result merging method, we compare the effects of different verifiers.

Diversifying Rollouts We explore this influence from two perspectives: on the one hand, we study the differences in agent capabilities under different sampling widths; on the other hand, we intro-

duce multi-agent rollouts to explore the benefits of increasing rollout diversity. To further illustrate the impact of diversifying rollouts, we introduce the pass@K metric, which measures the probability that at least one of K generated solutions for a problem is correct. This metric assesses the maximum capability a multi-agent collective can achieve when using different model compositions for its rollouts.

3.2 Baseline

We have compared the majority of existing agent frameworks(repository, 2022; Li, 2025; Liang et al., 2025; Hong et al., 2023) and selected Smolagents(Face, 2025) as our baseline due to its stability and superior performance. In this framework, agents take on different roles such as code actor and tool calling. In order to more intuitively compare the differences between various TTS algorithms, we remove the nesting of ToolAgent in the original smolagent framework and only use CodeAgent to directly call tools. We choose GPT-4.1(OpenAI, 2025) as the baseline model for the majority of our experiments. Additionally, we select current state-of-the-art models including Claude-3-7(Anthropic, 2025), Gemini-2.5-Pro(Google Cloud, 2025), and Claude-3-5(Anthropic, 2024) for comparative experiments involving mixed models.

3.3 Benchmark

We choose GAIA(Mialon et al., 2023) as our evaluation benchmark. The GAIA validation dataset contains 165 samples across three different difficulty levels. It primarily assesses agents' capabilities in web search and handling multimodal files.

4 Experimental Results

4.1 Comparison of Different Parallel Sampling Algorithms

Findings 1

The Parallel Sampling Algorithms significantly enhance agent performance.

As shown in Table 1, we present a comparison of using different parallel sampling algorithms against a baseline without any sampling algorithm. The experimental results demonstrate that by applying parallel sampling algorithms, agents can achieve superior performance. First, when comparing our baseline with the current open-source agent framework, our baseline achieved a score of 55.76, on par with the best existing framework, TapeAgent. FurthermoreCompared to the baseline, BoN, BoN-wise, and Beam-Search achieve significant performance gains, while DVTS performs similarly to the baseline. These results demonstrate the general effectiveness of Parallel Sampling Algorithms in the agentic framework. Meanwhile, different parallel sampling algorithms exhibit varying performance characteristics.

The BoN algorithm achieves the best performance gains, with an eight-point improvement over the baseline, and achieves SOTA results on level 1 and level 2. These two levels are heavily dependent on the agent's ability to call and use tools. Under the BoN algorithm, the agent is given more opportunities to repeatedly attempt similar tasks, which enhances performance particularly on simpler and mid-level difficult problems. BoN-wise achieves the second-best results after BoN, with a three-point improvement over the baseline. In particular, BoN-wise achieves the best performance on the most difficult level3 problems, surpassing both the baseline and BoN. BoN-wise allows for the largest exploration space at each decision node, further demonstrating that increasing step-wise exploration leads to better performance on complex tasks.

Notably, Beam-search and DVTS show no significant improvement over baseline. This is because although these algorithms can significantly increase the agent's exploration space, their exploration also depends on the accuracy of signals provided by the verify model, which prevents the agent from stably approaching the correct answers.

4.2 The impact of different Sequential Revision Strategies

Findings 2

Understanding the opportune moments for reflection is key to its profound benefit.

As shown in Table 2, we first compare the baseline agent with the reflection-enabled agent to assess the effectiveness of self-reflection. The baseline achieves an overall score of 55.76, while the reflection scores slightly lower at 55.15, suggesting that reflection, while enabling error correction, may also disrupt the model's reasoning flow. At Level

Table 1: **Comparison with Open-Source Agentic Models and Open-Source Agent Frameworks**. For the open-source models and frameworks, we adopt the results reported in their official papers. For our method, we consistently use GPT-4.1 as the base model for benchmarking.

Framework	Model Family	Average	Level 1	Level 2	Level 3
Agentic Model					
Search-o1-32B (Li et al., 2025a)	-	39.8	53.8	34.6	16.7
WebThinker-32B-RL (Li et al., 2025b)	-	48.5	56.4	50.0	16.7
Open-Source Agent Frameworks					
TapeAgents (Bahdanau et al., 2024)	Claude-3-7 *etc*	55.76	71.70	53.49	30.77
AutoAgent (Tang et al., 2025)	Claude-3-5 *etc*	55.15	71.70	53.40	26.92
Open Deep Research (AI, 2025)	OpenAI o1	55.15	67.92	53.49	34.62
Magnetic-1 (Fourney et al., 2024)	OpenAI o1 *etc*	46.06	56.60	46.51	23.08
FRIDAY (Wu et al., 2024c)	GPT-4 turbo	34.55	45.28	34.88	11.54
Smolagents (Face, 2025)	Openai o1 *etc*	53.33	62.26	54.65	30.77
Parallel Sampling Algorithms					
Baseline	GPT-4.1	55.76	66.04	58.14	26.92
BoN	GPT-4.1	**63.03**	**77.36**	**63.95**	30.77
BoN-wise	GPT-4.1	58.79	69.23	58.62	**38.46**
Beam-Search	GPT-4.1	56.97	69.81	55.81	34.62
DVTS	GPT-4.1	55.76	58.49	62.79	26.92

1, reflection significantly improves performance (71.7), indicating its benefit for simple tasks where minor errors can be quickly corrected. However, at Level 2, reflection underperforms compared to the baseline, suggesting that moderate-complexity tasks are more susceptible to disruption from excessive introspection. At Level 3, reflection leads to a moderate improvement (34.62), showing its value in preventing critical failures in complex scenarios.

Table 2: Performance with reflection

Search type	Score	Level 1	Level 2	Level 3
Baseline	55.76	66.04	58.14	26.92
Reflection	55.15	71.7	51.16	34.62
Threshold(<8)	53.33	66.04	53.49	26.92
Threshold(<5)	52.12	69.81	50.0	23.08
Threshold(<2)	56.36	71.7	55.81	26.92

As reflection introduces both benefits and potential disruptions, we examine how varying the frequency of reflection impacts task execution across different levels of complexity. Frequent reflection (threshold <8) results in the lowest overall score (53.66), particularly hurting performance at Level 2 due to reasoning interruptions. Moderate reflection (<5) yields even lower performance (52.12), whereas selective reflection (<2) achieves the best result (56.36), outperforming other strategies across all levels. This indicates that restricting reflection to only the most critical steps minimizes disruption while still allowing meaningful error correction. These findings suggest that effective use of reflection depends heavily on its application frequency and timing—low-frequency, context-aware reflection is most beneficial, especially for maintaining coherence in multi-step reasoning processes.

4.3 The impact of different verifiers and result merging methods

Findings 3

The list-wise approach outperforms alternative methods in both verification and result merging.

The impact of different result merging methods As shown in Table 3, we first compare common result merging methods. For these three algorithms - BoN, Beam-Search, and Tree Search - the list-wise approach outperforms other approaches. This is

because: 1) compared to scoring that directly relies on standard scoring, list-wise has comparable standards for reference, making evaluation more accurate; and 2) compared to voting methods, list-wise not only considers majority options in the answers, but can also select potentially correct answers from diverse candidates.

Table 3: Comparing performance of different result merging methods thought BoN and Beam-Search.

Search type	voting	Scoring	list-wise
BoN	56.8	59.39	63.03
Beam-Search	54.55	53.94	56.97

The impact of different verify methods As shown in Table 4, we compare the effects of different verification methods on agent performance. The list-wise verify method scores 3 points higher on average than the scoring, which indicates that, whether in BoN-wise or Beam-Search, using list-wise comparison of candidates is superior to the scoring approach. This suggests that, compared to directly having PRM score the agent's trajectory (scoring), using a list-wise approach to have PRM select the relatively optimal trajectory can bring more precise benefits.

Table 4: Performance with different verify methods across various search methods.

Search type	Verify method	Score
BoN-wise	scoring	56.36
	list-wise	58.79
Beam-Search	scoring	53.94
	list-wise	56.97
Tree-Search	scoring	50.91
	list-wise	55.76

4.4 The impact of Diversifying Rollouts

Findings 4

Increasing diversifying rollouts enhances agent performance.

Performance with different search size Figure 3 presents the performance variations of the agent under different sampling widths. The experimental results demonstrate that increasing the agent's sampling width leads to significant performance improvements, a finding that aligns with test-time scaling phenomena observed in the LLM domain.

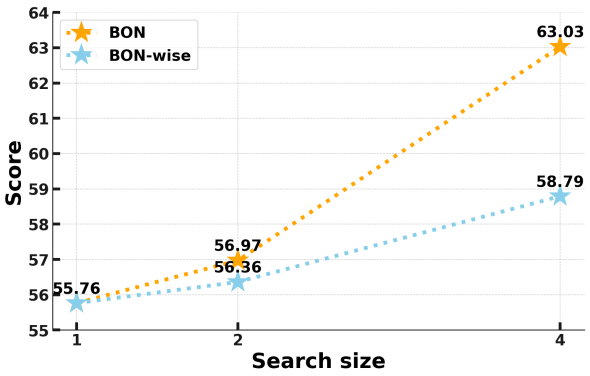

Figure 3: Performance comparison of BoN and BoN-wise algorithms across different search sizes.

Performance with different rollout models To further explore the impact of different rollout models on Diversifying Rollouts, we measure the pass@1 performance of SOTA models and calculate their pass@2 and pass@4 performance when combined with the baseline model GPT-4.1.

As shown in Table 5, under the pass@1 setting, GPT-4.1 achieves the highest score. Notably, under the pass@2 and pass@4 settings, using other models mixed with GPT-4.1 yields higher results than using GPT-4.1 alone, which further demonstrates the effectiveness of using different rollouts models. Additionally, using all four different models achieves a total score of **74.55** for pass@4, reaching a level that surpasses the open-source SOTA.

5 Related Work

Language Agents In previous research work, many mature agentic frameworks have been established, such as Meta-GPT(Hong et al., 2024) which enables GPT to assume different roles and collaboratively complete tasks, LangChain(repository, 2022) which uses natural language to describe tools and solve complex tasks, and AGENTS(Zhou et al., 2023c) which, in addition to supporting basic tool calling and long-term memory, also supports human-agent interaction and controllability through symbolic plans (SOPs). Recently, with the emergence of OPENAI's deep research concept, numerous agentic frameworks have appeared, such as Manus(Liang et al., 2025), OWL(Li, 2025), and SmoLAgents(Face, 2025). These agentic frame-

Table 5: Performance with different rollout models evaluated by Pass@K. Under each setting, the best performance is indicated with underlining.

Model	all	level1	level2	level3
GPT-4.1	55.76	66.04	58.14	26.92
Claude-3-5	42.42	50.94	46.51	11.54
Claude-3-7	50.30	54.72	50.00	42.31
Gemini-2.5-PRO	41.82	54.72	41.86	15.38
Pass@2				
GPT-4.1 only	60.49	70.59	60.00	42.31
GPT-4.1,Claude-3-5	64.24	71.70	69.77	30.77
GPT-4.1,Claude-3-7	64.24	71.70	63.95	50.00
GPT-4.1,Gemini-2.5-PRO	62.42	79.25	60.47	34.62
Pass@4				
GPT-4.1 only	69.14	82.35	71.76	34.62
GPT-4.1,Claude-3-5, Gemini-2.5-PRO,Claude-3-7	74.55	86.79	74.42	50.00

works support collaborative work among various agents, decompose task inputs, conduct multi-step task planning, and invoke diverse tools to complete complex tasks. However, current agentic frameworks predominantly employ a single linear workflow to solve problems and have not yet conducted an in-depth exploration of Test-Time-Scaling (TTS) capabilities.

LLM Test-Time Scaling (Snell et al., 2024) propose that scaling LLMs Test-time Compute (Wu et al., 2025; Yu et al., 2025) optimally can be more effective than scaling model parameters. OpenAI's o1 model[1] is designed to spend more time reasoning before they respond for the sake of obtaining better performance. Recently, various TTS algorithms have emerged, such as Best-Of-N, Beam-Search, Tree-Search, and Majority-Vote (Liu et al., 2025c; Faria and Smith, 2025; Koh et al., 2024b). Moreover, (Kumar et al., 2024) and (Xiong et al., 2025) investigate enabling LLMs to perform self-reflection through self-rewarding and self-correlation mechanisms to enhance their performance.

Besides, Some works(Snell et al., 2024; Wu et al., 2024b; Beeching et al., 2024b) design reward models to guide the trajectory selection process in LLM test-time scaling . (Chen et al., 2025) incorporates deep thinking into reward models, while (Qian et al., 2025) uses RM for complex tool selection. (Liu et al., 2025b) and (Wu et al., 2024a) provide a comprehensive experimental analysis of the LLM test-time scaling. However, test-time scaling strategies have not yet been fully discussed within agentic frameworks.

In this work, we investigate four key aspects of test-time scaling strategies: parallel sampling algorithms, sequential revision strategies, verifiers and merging methods, and diversifying rollouts, and conduct comprehensive ablation experiments comparing various strategies within each aspect.

6 Conclusion

Test Time Scaling (TTS) can significantly enhance LLM inference performance by increasing computational resources during the inference phase. However, the application of Test Time Scaling in the agentic domain still needs to be explored. In this paper, we explore four different aspects of test-time scaling strategies: Parallel Sampling Algorithms; Sequential Revision Strategies; Verifiers and Merging Methods; Strategies for Diversifying Rollouts. We conduct detailed comparative ablation experiments for strategies in each aspect. Our experimental results indicate: 1. Applying parallel sampling algorithms to scale agent test time compute could improve agent performance; 2. For sequential revision, it is important to know when to revise; 3. Among different verify and result merge methods; list-wise methods perform best; 4. Increasing diversified rollouts exerts a positive benefit on agent, which aligns with test-time scaling phenomena observed in the LLM domain.

[1] https://openai.com/o1/

Limitations

In this work, due to resource constraints, we do not investigate the test-time scaling capabilities of open-source models within language agents. This work employs multiple LLMs to implement BoN (N=4), but how to more effectively expand the search space remains an open question for future research.

Ethics Statement

The dataset used in our research is constructed using publicly available data sources, ensuring that there are no privacy concerns or violations. We do not collect any personally identifiable information, and all data used in our research is obtained following legal and ethical standards.

References

LangChain AI. 2025. Open deep research: A fully open-source research assistant.

Anthropic. 2024. Claude 3.5 sonnet model card addendum. Accessed: 2024-09-21.

Anthropic. 2025. Claude 3.7 Sonnet and Claude Code.

Dzmitry Bahdanau, Nicolas Gontier, Gabriel Huang, Ehsan Kamalloo, Rafael Pardinas, Alex Piché, Torsten Scholak, Oleh Shliazhko, Jordan Prince Tremblay, Karam Ghanem, Soham Parikh, Mitul Tiwari, and Quaizar Vohra. 2024. Tapeagents: a holistic framework for agent development and optimization. *Preprint*, arXiv:2412.08445.

Edward Beeching, Lewis Tunstall, and Sasha Rush. 2024a. Scaling test-time compute with open models. Accessed: [Current Date, e.g., 28 July 2025].

Edward Beeching, Lewis Tunstall, and Sasha Rush. 2024b. Scaling test-time compute with open models.

Xiusi Chen, Gaotang Li, Ziqi Wang, Bowen Jin, Cheng Qian, Yu Wang, Hongru Wang, Yu Zhang, Denghui Zhang, Tong Zhang, and 1 others. 2025. Rm-r1: Reward modeling as reasoning. *arXiv preprint arXiv:2505.02387*.

Hugging Face. 2025. Smolagent: A scalable approach to multi-agent systems.

Gonçalo Faria and Noah A. Smith. 2025. Sample, don't search: Rethinking test-time alignment for language models. *Preprint*, arXiv:2504.03790.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang (Eric) Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. 2024. Magentic-one: A generalist multi-agent system for solving complex tasks.

Google Cloud. 2025. Gemini 2.5 Pro | Generative AI on Vertex AI.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, and 1 others. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 3(4):6.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. *Preprint*, arXiv:2308.00352.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. 2024a. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. 2024b. Tree search for language model agents. *Preprint*, arXiv:2407.01476.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, and 1 others. 2024. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*.

Mengkang Hu Yuhang Zhou Wendong Fan Yuzhou Nie Bowei Xia Tao Sun Ziyu Ye Zhaoxuan Jin Yingru Li Zeyu Zhang Yifeng Wang Qianshuo Ye Ping Luo Guohao Li. 2025. Owl: Optimized workforce learning for general multi-agent assistance in real-world task automation.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025a. Search-o1: Agentic search-enhanced large reasoning models. *Preprint*, arXiv:2501.05366.

Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yutao Zhu, Yongkang Wu, Ji-Rong Wen, and Zhicheng Dou. 2025b. Webthinker: Empowering large reasoning models with deep research capability. *Preprint*, arXiv:2504.21776.

Xinbin Liang, Jinyu Xiang, Zhaoyang Yu, Jiayi Zhang, Sirui Hong, Sheng Fan, and Xiao Tang. 2025. Openmanus: An open-source framework for building general ai agents.

Fan Liu, Wenshuo Chao, Naiqiang Tan, and Hao Liu. 2025a. Bag of tricks for inference-time computation of llm reasoning. *Preprint*, arXiv:2502.07191.

Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, Biqing Qi, Wanli Ouyang, and Bowen Zhou. 2025b. Can 1b llm surpass 405b llm? rethinking compute-optimal test-time scaling. *Preprint*, arXiv:2502.06703.

Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, Biqing Qi, Wanli Ouyang, and Bowen Zhou. 2025c. Can 1b llm surpass 405b llm? rethinking compute-optimal test-time scaling. *arXiv preprint arXiv:2502.06703*.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*.

OpenAI. 2025. Introducing GPT-4.1 in the API.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*.

Langchain repository. 2022. Langchain.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Jiabin Tang, Tianyu Fan, and Chao Huang. 2025. Autoagent: A fully-automated and zero-code framework for llm agents. *Preprint*, arXiv:2502.05957.

Jun Wang, Meng Fang, Ziyu Wan, Muning Wen, Jiachen Zhu, Anjie Liu, Ziqin Gong, Yan Song, Lei Chen, Lionel M Ni, and 1 others. 2024. Openr: An open source framework for advanced reasoning with large language models. *arXiv preprint arXiv:2410.09671*.

Siwei Wu, Zhongyuan Peng, Xinrun Du, Tuney Zheng, Minghao Liu, Jialong Wu, Jiachen Ma, Yizhi Li, Jian Yang, Wangchunshu Zhou, and 1 others. 2024a. A comparative study on reasoning patterns of openai's o1 model. *arXiv preprint arXiv:2410.13639*.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024b. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2025. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *Preprint*, arXiv:2408.00724.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. 2024c. Os-copilot: Towards generalist computer agents with self-improvement. *Preprint*, arXiv:2402.07456.

Wei Xiong, Hanning Zhang, Chenlu Ye, Lichang Chen, Nan Jiang, and Tong Zhang. 2025. Self-rewarding correction for mathematical reasoning. *arXiv preprint arXiv:2502.19613*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Zhouliang Yu, Yuhuan Yuan, Tim Z Xiao, Fuxiang Frank Xia, Jie Fu, Ge Zhang, Ge Lin, and Weiyang Liu. 2025. Generating symbolic world models via test-time scaling of large language models. *arXiv preprint arXiv:2502.04728*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2023a. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

Wangchunshu Zhou, Yuchen Eleanor Jiang, Peng Cui, Tiannan Wang, Zhenxin Xiao, Yifan Hou, Ryan Cotterell, and Mrinmaya Sachan. 2023b. Recurrentgpt: Interactive generation of (arbitrarily) long text. *Preprint*, arXiv:2305.13304.

Wangchunshu Zhou, Yuchen Eleanor Jiang, Long Li, Jialong Wu, Tiannan Wang, Shi Qiu, Jintian Zhang, Jing Chen, Ruipu Wu, Shuai Wang, Shiding Zhu, Jiyu Chen, Wentao Zhang, Xiangru Tang, Ningyu Zhang, Huajun Chen, Peng Cui, and Mrinmaya Sachan. 2023c. Agents: An open-source framework for autonomous language agents. *Preprint*, arXiv:2309.07870.

Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen, Shuai Wang, Xiaohua Xu, Ningyu Zhang, Huajun Chen, and Yuchen Eleanor Jiang. 2024. Symbolic learning enables self-evolving agents.

A Appendix

PRM-score Evaluation Prompt

Evaluation Guidelines:

- **Objective:**
  - You will evaluate a candidate ActionStep node, which includes the following fields:
    - `step_number`: Depth of this step within the TTS search tree.
    - `observations`: Observations recorded after executing this action.
    - `action_output`: Direct output resulting from this action.
    - `model_output`: Raw LLM output that led to this action.
    - `error`: Any encountered errors (can be None).
    - `score`: Previously assigned score (for reference only).
    - `previous_steps`: The history of earlier steps, including TaskStep and PlanningStep, along with the trajectory of ActionSteps leading to the current state.
  - Your goal is to judge how promising this ActionStep is for advancing toward the user's task, using your independent judgment while considering the continuity and logical flow of the ActionStep sequence, including the historical context.
- **Evaluation Criteria:**
  - **Progress Toward Goal:**
    - Assess whether the `action_output` clearly and tangibly advances the overall task.
    - Reward meaningful progress or valuable new information.
    - Penalize irrelevant actions or weak impact.
  - **Error and Stability:**
    - Penalize based on the severity of errors:
      - Fatal/blocking errors: 0-1 points.
      - Significant errors: 1-3 points.
      - Minor or recoverable errors: 3-5 points.
    - Reduce the score if the `model_output` is ambiguous or unstable.
  - **TTS Efficiency:**
    - Reward actions that contribute efficiently toward reaching the goal.
    - Penalize redundant or repetitive actions without meaningful progress.
  - **Reflection Usage:**
    - Reward active utilization of `reflection` to improve upon past mistakes.
    - Penalize ignoring reflection insights.
  - **Loop Detection:**
    - Detect loops or repetitions compared to previous steps.
    - Identify true loops and penalize based on severity.
  - **Contextual Awareness:**
    - Infer alignment with previous `PlanningStep` and `TaskStep`.
    - Ensure consistency with the TTS strategy and penalize deviations.
- **Scoring Criteria:**
  - `9-10`: Clearly advances the goal; highly efficient; strong reflection use; no loops.
  - `7-8`: Good advancement; minor inefficiencies; clear reflection use; minimal loop risk.
  - `5-6`: Moderate progress; limited efficiency; moderate reflection use; mild repetition risks.
  - `3-4`: Poor advancement; inefficient; weak reflection use; noticeable loop risks.
  - `1-2`: Minimal advancement; repetitive actions; true loops; significant errors.
  - `0`: Severe issues: explicit loops, critical errors, or complete irrelevance to the task context.
- **Final Evaluation Output:** You must provide your evaluation in valid JSON format with the following structure:

```
{ "analysis": "Detailed analysis addressing progress, TTS efficiency, reflection
usage, loop detection, contextual alignment with PlanningStep/TaskStep, error
severity, and overall action quality.", "score": [integer between 0-10] }
```

PRM-list Evaluation Prompt

Evaluation Guidelines:

- **Objective:**
  - You will evaluate N candidate trajectories, each representing a series of nodes in a search tree. Each trajectory contains the following:
    - `step_number`: Depth of the node in the trajectory.
    - `observations`: Observations recorded at each step of the trajectory.
    - `action_output`: Direct action output at each step.
    - `model_output`: Raw model output (LLM).
    - `error`: Any errors encountered (can be None).
    - `score`: Previously calculated score (if available).
    - `previous_steps`: The history of earlier steps, including TaskStep and PlanningStep, with the trajectory of ActionSteps leading to the current state.
  - Your goal is to evaluate each trajectory holistically, considering how well it progresses toward solving the user's task. Select the trajectory that most effectively achieves this goal.
- **Evaluation Criteria:**
  - **Progress Toward Goal:**
    - Assess how well each trajectory advances the task at hand, considering both the individual node's progress and the overall progression of the entire trajectory.
    - Reward trajectories that demonstrate tangible and meaningful progress toward the goal.
    - Penalize trajectories with weak actions or minimal/no advancement.
  - **Trajectory Efficiency:**
    - Evaluate how efficiently each trajectory progresses toward the goal, considering the depth and complexity of the steps.
    - Favor trajectories that achieve significant progress with fewer steps.
    - Consider the overall value-to-depth ratio when comparing trajectories of different lengths.
    - Reward efficient exploration of the search space.
  - **Loop Detection:**
    - Detect loops or repetitions within each trajectory, especially those related to previous steps.
    - **Loop types:**
      - `Real Loops`: Identical nodes (observations, action output, and model output) that do not add value to the trajectory.
      - `Benign Repetitions`: Similar strategies with variations yielding additional progress.
    - Heavily penalize trajectories with real loops.
    - Slight penalties for benign repetitions if they lead to meaningful improvements.
  - **Error and Stability:**
    - Evaluate the severity of errors encountered in each trajectory and penalize based on their impact on progression.
    - **Error Severity:**
      - Fatal/Blocking Errors: Major penalty.
      - Significant Errors: Moderate penalty.
      - Minor/Recoverable Issues: Minor penalty.
    - Penalize unstable or unclear model outputs.
    - Consider how errors affect the overall trajectory's ability to move toward the goal.
  - **Overall Trajectory Quality:**
    - Evaluate the coherence and overall quality of the trajectory.
    - Consider the logical sequence of steps and the exploration-exploitation balance.
    - Evaluate the final node's closeness to achieving the goal.
    - Reward trajectories that make consistent progress and demonstrate coherent planning.
- **Final Output Format:** Provide your evaluation in the following JSON format. Select the best trajectory and provide a detailed analysis explaining why it is the most promising trajectory.

```
{ "index": [integer], # Index of the best trajectory "analysis": "Detailed
analysis addressing progress, efficiency, reflection usage, loop detection, error
severity, and overall trajectory quality." }
```

Single Node Reflection Prompt

Node Information:

- `step_number`: The depth of the node within the BON/beam search tree.
- `observations`: The data or observations recorded during this step.
- `action_output`: The direct output resulting from an action taken at this step (e.g., API call, tool response).
- `model_output`: The raw output generated by the model at this step.
- `error`: Any errors encountered during this step (if applicable).

Goal:

- **Summarize:**
  - Provide a brief overview of what occurred at this node.
  - Describe the action taken and the results or new information that emerged as a result of this action.
- **Reflect:**
  - Assess whether the action taken in this node was successful, partially successful, or unsuccessful.
  - Identify any errors, issues, or incompleteness relevant to this step.
  - Compare the node's outcome with its assigned score, providing an evaluation of whether the score is aligned with the actual result.
- **Confidence:**
  - Evaluate your confidence in the action taken at this node (High/Medium/Low).
  - If confidence is high, explicitly suggest continuing along this exploration path.
  - If confidence is medium or low, recommend potential improvements or alternatives, while leaving room for exploration to remain open.
- **Suggest:**
  - Provide specific and focused suggestions for refining the current step.
  - These should be based on the evaluation of the current node, with an emphasis on actionable changes that can be made in the next attempt of a similar step.
  - Focus exclusively on improvements that can be applied within this node. Avoid proposing changes that span multiple steps or introduce larger, long-term strategies.
  - Base your evaluation strictly on the provided fields—action_output, observations, error, etc. Do not infer additional context or hypothesize about alternative paths or unknown factors.
  - Only flag a step as unsuccessful or in need of improvement if there is clear, tangible evidence (e.g., explicit errors, missing or incorrect outputs).
  - Do not override factual results based on subjective judgment, even if the node's score does not seem to match the outcome.
- **General Guidelines:**
  - Your suggestions should be conservative, focusing only on changes where there is a clear issue or opportunity for improvement.
  - If no significant issues are identified, provide minimal or no suggestions for improvement.

Output Format:

- **experience_summary**: A concise overview of the events at this node and the key outcomes.
- **confidence_assessment**: High/Medium/Low with a recommendation for future exploration.
- **lessons_learned**: Key takeaways or specific improvements based on the evaluation of the current node's action.
- **comments**: Optional minor remarks, clarifications, or additional observations.